PROBLEM SOLVING AND PYTHON PROGRAMMING

UNIT 1

Unit - 1

COMPUTATIONAL THINKING AND PROBLEM SOLVING

Fundamentals of Computing – Identification of Computational Problems -Algorithms, building blocks of algorithms (statements, state, control flow, functions), notation (pseudo code, flow chart, programming language), algorithmic problem solving, simple strategies for developing algorithms (iteration, recursion). Illustrative problems: find minimum in a list, insert a card in a list of sorted cards, guess an integer number in a range, Towers of Hanoi.

FUNDAMENTALS OF COMPUTING

- Computers are seen everywhere around us, in all spheres of life, in the field of education, research, travel and tourism, weather forecasting, social networking, e-commerce etc. Computers have now become an essential part of our lives. Today, no organization can function without a computer. In fact, various organizations have become paperless. Computers have evolved over the years from a simple calculating device to high-speed portable computers.

- A computer is an electronic device that can be programmed to accept data (input), process it and generate result (output). A computer along with additional hardware and software together is called a computer system.

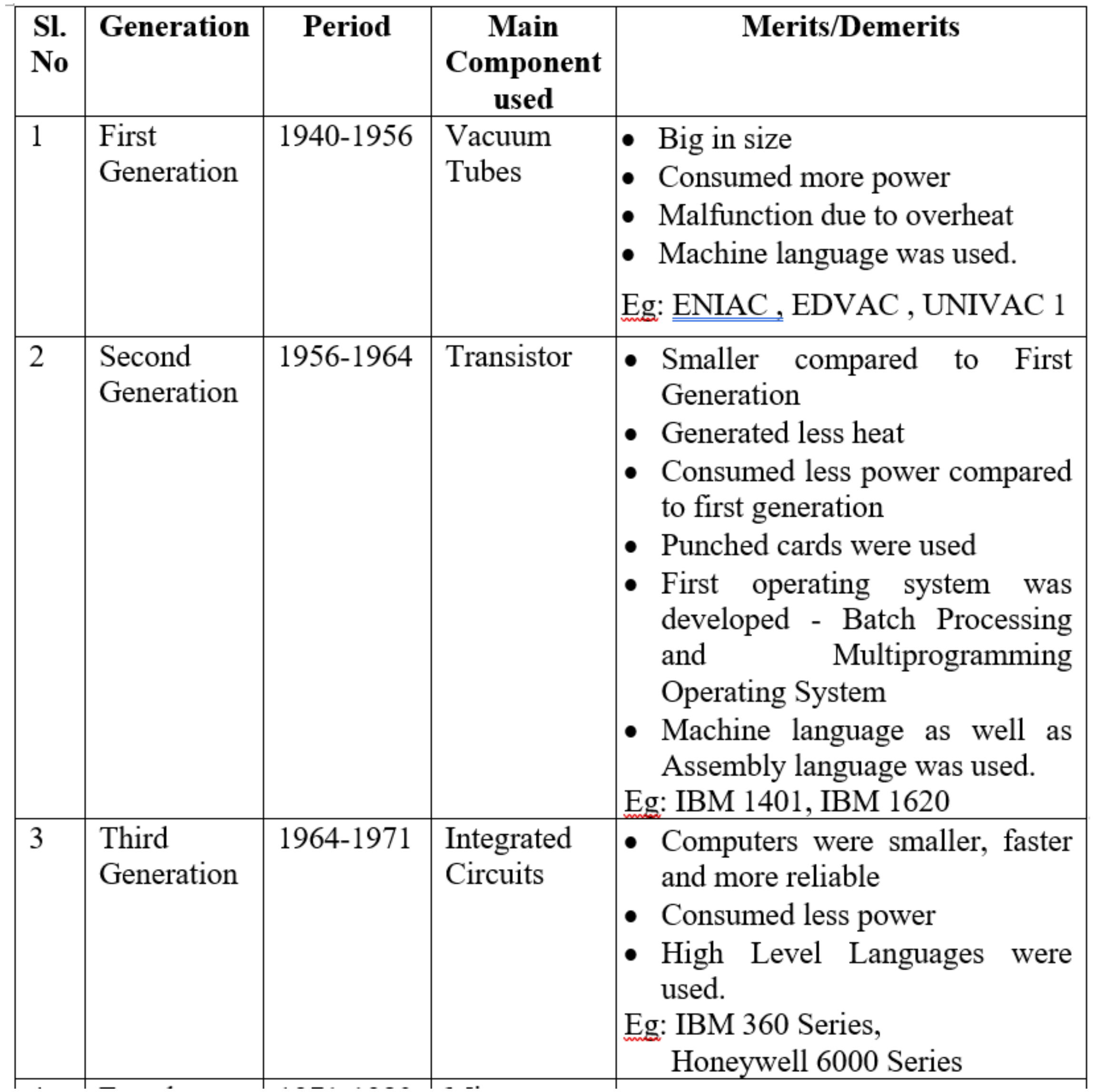
- Computers are very versatile as they do a lot of different tasks such as storing data, weather forecasting, booking airlines, railway or movie tickets and even playing games.

- **Data:** Data is defined as an un-processed collection of raw facts, suitable for communication, interpretation or processing.

- **Information:** Information is a collection of facts that is processed to give meaningful, ordered or structured information.

Generations of a Computer

- Growth in the computer industry is determined by the development in technology. Based on various stages of development, computers can be categorized into different generations.



Sl. No	Generation	Period	Main Component used	Merits/Demerits
1	First Generation	1940-1956	Vacuum Tubes	• Big in size • Consumed more power • Malfunction due to overheat • Machine language was used. Eg: ENIAC , EDVAC , UNIVAC 1
2	Second Generation	1956-1964	Transistor	• Smaller compared to First Generation • Generated less heat • Consumed less power compared to first generation • Punched cards were used • First operating system was developed - Batch Processing and Multiprogramming Operating System • Machine language as well as Assembly language was used. Eg: IBM 1401, IBM 1620
3	Third Generation	1964-1971	Integrated Circuits	• Computers were smaller, faster and more reliable • Consumed less power • High Level Languages were used. Eg: IBM 360 Series, Honeywell 6000 Series

4	Fourth Generation	1971-1980	Micro Processors	• Smaller and Faster • Microcomputer series such as IBM and APPLE were developed • Portable Computers were introduced.
5	Fifth Generation	1980 – Till Date	Ultra-Large-Scale Integration	• Parallel Processing • Super conductors • Computer's size was drastically reduced. • Can recognise Images and Graphics • Introduction of Artificial Intelligence and Expert Systems • Able to solve high complex problems including decision making and logical reasoning.
6	Sixth Generation	In Future		• Parallel and Distributed computing • Computers have become smarter, faster and smaller • Development of robotics • Natural Language Processing • Development of Voice Recognition Software

Components of a Computer

The computer is the combination of hardware and software. Hardware is the physical component of a computer like motherboard, memory devices, monitor, keyboard etc., while software is the set of programs or instructions. Both hardware and software togethe
function.

- The computer system hardware comprises of three main components

1. Input / Output (I/O) Unit,

2. Central Processing Unit (CPU), and

3. Memory Unit

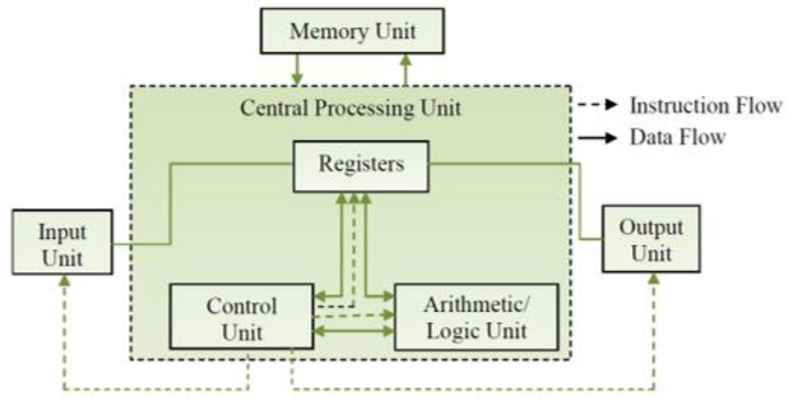


Fig. 1.1 The Computer System Interaction

Input / Output Unit

The input/output unit consists of the Input unit and the Output unit.

The user interacts with the computer via the I/O unit.

The **Input unit**

It accepts data from the user.
It converts the data into a form that is understandable by the computer.

The input is provided to the computer using devices like keyboard, trackball and mouse.

The **Output unit**

- provides the processed data (i.e.) the information to the user.
- provides output in a form that is understandable by the user.
- Some of the output devices are monitor and printer.

Central Processing Unit: (CPU)

- CPU or the processor is often called the Brain of computer.
- CPU controls, coordinates and supervises the operations of the computer.
- CPU consists of

1. Arithmetic Logic Unit (ALU)
2. Control Unit (CU) and
3. Set of Registers.

Arithmetic Logic Unit

- ALU consists of two units. They are arithmetic unit and logic unit.
- The **Arithmetic unit** performs arithmetic operations on the data that is made available to it. Example: addition, subtraction, multiplication and division.
- The **Logic unit** performs logic operations. Example: comparison of numbers, letters and special characters. Testing for greater than, less than or equal to condition.
- ALU performs arithmetic and logic operations, and uses registers to hold the data that is being processed.

Registers (contd..)

- Registers are high-speed storage areas within the CPU, but have the least storage capacity.
- Registers are not referenced by their address, but are directly accessed and manipulated by the CPU during instruction execution.
- Registers store data, instructions, addresses and intermediate results of processing. So, registers are often called as **CPU’s working memory**.
- The data and instructions that require processing must be brought in the registers of CPU before they can be processed.
- For example, if two numbers are to be added, both numbers are brought in the registers, added and the result is again placed in the register.

Registers

Some of the important registers in CPU are as follows.

ACC: Accumulator stores the result of arithmetic and logic operations.

IR: Instruction Register contains the current instruction most recently fetched.

PC: Program Counter contains the address of next instruction to be processed.

MAR: Memory Address Register contains the address of next location the in memory to be accessed.

MBR: Memory Buffer Register temporarily stores data from memory or the data to be sent to memory.

DR: Data Register stores the operands and any other data.

The size of register, also called **Word Size**, indicates the amount of data. The size of a register may be 8, 16, 32 or 64 bits.

Control Unit: (CU)

- The control unit organizes the processing of data and instructions.
- The CU acts as a supervisor, controls and coordinates the activity of the other units of computer.
- CU coordinates the input and output devices of a computer.
- It directs the computer to carry out stored program instructions in the IR.
- It instructs the ALU to perform the arithmetic or logic operations.
- When a program is run, the PC keeps track of the instruction to be executed next.
- CU tells ‘when to fetch the data and instructions’, ‘what to do’, ‘where to store the result’ etc.
- CU also holds the CPU’s Instruction Set, which is a list of all operations that the CPU can perform.

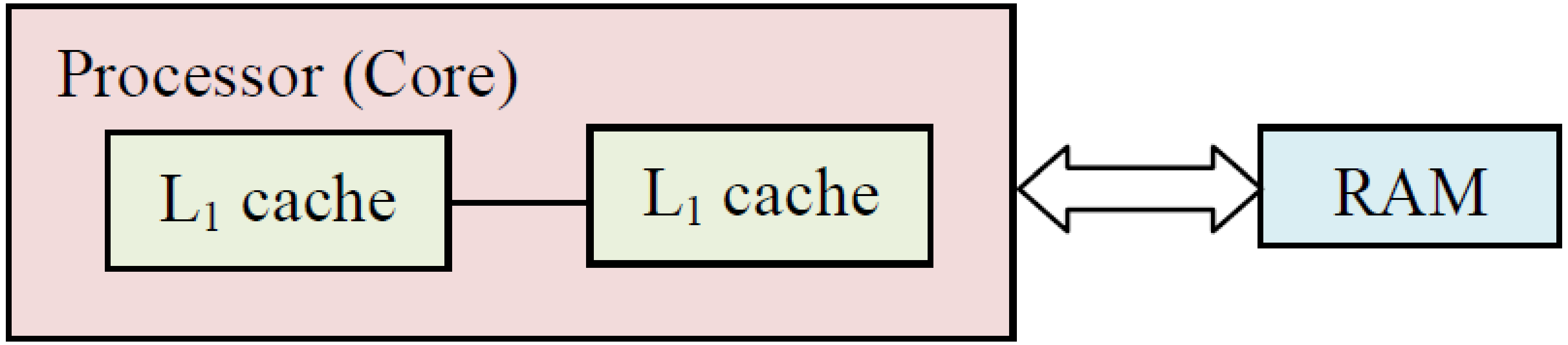
Memory Unit

- The memory unit consists of cache memory, primary memory and secondary memory.
- Memory unit stores the data and instructions, intermediate results and output temporarily during processing of data.
- This memory is called main memory / primary memory. Example: RAM, ROM.

Cache Memory

- The data and instructions that are required during the processing of data are brought from the secondary storage devices and stored in the RAM.
- Next for processing they are taken from RAM to registers.
- The time taken to move the data between RAM and CPU registers is large. This affects the speed of processing of computer and results in decreasing the performance of CPU.
- Cache memory (high speed memory) is placed in between RAM and CPU.
- Cache memory is a storage buffer that stores the data that is used more often. So, during processing, CPU first checks cache memory for required data. If data is not found in cache, then it looks in RAM for that data.
- To access the cache memory, CPU does not have to use the mother board's system bus for data transfer.
- Cache is twice as fast as RAM.
- The CPU has a built-in Level 1 (L1) cache and Level 2 (L2) cache, as shown in below.

Pictorial Representation



Primary Memory

- Primary memory is the main memory of computer.
- It is used to store data and instructions during the processing.
- It is semiconductor memory.
- Primary memory is of two kinds

    (i) Random Access Memory (RAM)

    (ii) Read Only Memory (ROM).

RAM

- RAM is volatile.
- It stores data when the computer is on.
- Information in RAM gets erased when the computer is turned off.
- RAM provides temporary storage for data and instructions.
- It provides a limited storage capacity, due its high cost.

ROM

- ROM is non-volatile memory and is a read only memory.
- Storage in ROM is permanent in nature.
- Information will not be erased even if the computer is turned off.
- ROM comes programmed by the manufacturer.

Secondary Memory

- The secondary memory stores data and instructions permanently. The information can be stored in secondary memory for a long time (years), and is generally permanent in nature unless erased by the user.

- Non-volatile memory.
- It provides back-up storage for data and instructions. Hard disk drive, floppy drive and optical disk drives are some examples of secondary storage devices.
- The data and instructions that are currently not being processed, but may be required later by CPU are stored in secondary memory.
- Secondary memory has a high storage capacity than the primary memory.
- Secondary memory is also cheaper than the primary memory.
- But it takes longer time to access the data and instructions stored in secondary memory than in primary memory.

IDENTIFICATION OF COMPUTATIONAL PROBLEMS

- Identification of computational problem is part of the scientific method, as it serves as the first step in a systematic process to identify, evaluate a problem and explore potential solutions.
- Computational problem identification consists of two steps:

1. **Identifying and acknowledging that there is a problem**
2. **Developing a problem identification statement.**

- Computational problem Identification, require a certain mode of approach or way of thinking. This approach is often called **computational thinking** and is similar to the **scientific method with predictions**.
- Understanding computational thinking will give a foundation for solving problems. Computational thinking is part of a problem-solving process **that ends with writing a program**.
- In order to make predictions using computational thinking, we need to **define four steps** related to the identification of problem and its solution: They are, **decomposition, pattern recognition, abstraction, and algorithm design**.

Decomposition

- The first step of computational thinking is decomposition. This stage starts by analyzing the problem, stating it precisely, and establishing the criteria for the solution.

- A computational thinking approach to a solution **often starts by breaking the problem down into smaller more familiar components so they can be managed eas**ier. The more you can break a problem down, the easier it is to solve.

- This stage also allows to develop a better understanding of the problem by identifying all the components in detail.

Pattern Recognition

- The second step is pattern recognition whereby similarities and trends are identified within the problem.
- If some problems are similar in nature, there is a good chance that they **can be solved using similar, or repeated techniques.**
- This is a key component for making efficient solutions, and saving time in the long run. Pattern recognition is **a key skill to create efficient and effective solutions** to given problems.

Abstraction

- The abstraction stage involves the **identification of key components of the solution**.
- Rather than looking at specific details, it requires **the ability to filter out unnecessary elements of a problem so that only focus on the important elements**.
- Abstraction allows to consider all the key components prior to the creation of the final solution, while **ignoring any unnecessary detai**ls.

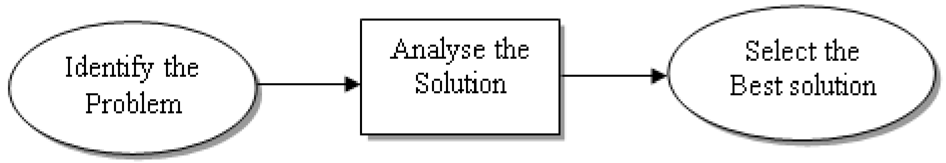
Algorithm Design

- **The final stage within the computational thinking process is algorithm design** whereby a detailed step-by-step set of instructions are created which explain how to solve the problem.
- The measure of a good algorithm is one that **can be passed to someone else to follow without the need for any extra explanation.**
- This process uses inductive thinking and is needed for transferring a particular problem to a larger class of similar problems. This step is also sometimes called algorithmic thinking.

NEED FOR LOGICAL ANALYSIS AND THINKING

- Computer can perform variety of tasks like receiving data, processing it and producing useful results. It cannot perform on its own. A computer needs to be instructed to perform a task.

- The computer works on set of instructions, called computer program to carry out a particular task, so that the computer can process that task.



ALGORITHMS

- Algorithm is an **ordered sequence** of finite, well defined, unambiguous instructions for completing a task.
- It is a **step-by-step procedure** for solving any problem.
- Algorithm is an **English-like representation of the logic** which is used to solve the problem.
- To accomplish a particular task, **different algorithms can be written**. They differ by their time and space.
- The programmer selects the best suited algorithm for the given task to be solved.
- The algorithm can be implemented in many different languages by using different methods and programs.
- The algorithm is **independent of any programming language**.

Guidelines for writing Algorithms

- ❑An algorithm should be **clear, precise and well defined**.
- ❑It should always begin with the word **'Start'** and end with the word **'Stop'.**
- ❑Each step should be written in a **separate line**.
- ❑Steps should be **numbered** as Step 1, Step 2, and so on.

Example: Algorithms to find the greatest among three numbers

Step 1:Start.

Step 2:Read the three numbers A, B, C.

Step 3:Compare A and B. If A is the greatest perform step 4 else perform step 5.

Step 4:Compare A and C. If A is the greatest, output “A is the greatest” else output “C is the greatest”.

Step 5:Compare B and C. If B is the greatest, output “B is the greatest” else output “C is the greatest”.

Step 6:Stop.

Properties of an Algorithm

- **Finiteness:** An algorithm must be **terminated** after a finite number of steps.
- **Definiteness:** Each step of an algorithm must be **precisely defined**.
- **Input:** The data must be present before any operations can be performed on it. The initial data is supplied by a READ instruction. A variable can be given initial value using the SET instruction.

  Example: READ A, B

  SET N=0
- **Output:** After executing all the steps of the algorithm at least one **output must be obtained.** The WRITE statement is used to print messages and variables.
- **Effectiveness:** The operations to be performed in the algorithm can be carried out manually in **finite intervals of time**.

Advantages of Algorithm

- It is a simple to understand step by step solution of the problem.
- It is easy to debug.
- It is independent of programming languages.
- It is compatible to computers, because each step of an algorithm can be easily coded into its equivalent high level language.

BUILDING BLOCKS OF ALGORITHMS

- The building blocks are,
  - Statements
  - State
  - Control flow and
  - Functions

Statements / Instructions

- ❑The algorithm consists of finite number of statements.
- ❑The statements must be in an ordered form.
- ❑The time taken to execute all the statements of the algorithm should be finite and within a reasonable limit.

State

- ❑The algorithm must have an instruction for a program to do.
- ❑The state of the instruction will be changed during execution.
- ❑The state or condition of the instruction determines the logic flow of the program or program termination.
- ❑During execution of the instruction the state will be stored in the data structure.

Control flow

- The logic of a program may not always be executed in a particular order. The execution of statements is based on a decision.

- The basic control flows needed for writing good and efficient algorithms are,
  - Sequence Control Flow
  - Selection Control Flow
  - Iteration (Looping) Control Flow

- These control flows allow the program to make choices, change direction or repeat actions.

Sequence Control Flow

- In sequence control flow, the instructions are executed in a linear order one after the other. The instructions in sequence control flow are executed exactly once.
- **Example:** Algorithm to find the sum of two numbers.
  - Step 1: Start
  - Step 2: Read two numbers A and B
  - Step 3: Calculate sum = A + B
  - Step 4: Print the sum value
  - Step 5: Stop.
- This algorithm performs the steps in a purely sequential order.

Selection Control Flow

- In a selection control flow, the step to be executed next is based on a decision taken. If the condition is true, one path is followed. If the condition is false, another path is followed.

Example: Algorithm to find the greatest among three numbers.

```
Step 1:   Start.
Step 2:   Read the three numbers A, B, C.
Step 3: Compare A and B. If A is the greatest perform step 4 else perform step 5.
Step 4:   Compare A and C. If A is the greatest, print "A is the greatest" else print "C is the greatest".
Step 5:   Compare B and C. If B is the greatest, print "B is the greatest" else print "C is the greatest".
Step 6:   Stop.
```

Iteration (Looping) Control Flow

- In iterative control flow, one or more instructions are executed repeatedly. In this control flow a condition is checked. Based on the result of this conditional check (true or false) the control goes back to one of the already executed steps to make a loop.
- **Example:** Algorithm to find the sum of first 100 integers.
- 1. Start
- 2. Assign sum = 0, i = 0.
- 3. Calculate i = i + 1 and sum = sum + i
- 4. Check whether i>= 100, if no repeat step 3. Otherwise go to next step.
- 5. Print the value of sum
- 6. Stop

Functions

- Any complex problem will become simpler if the problem is broken smaller and the smaller problems are solved. **The functions are solutions to smaller problems. These can be used to solve bigger, complex problems**.
- During algorithm design the **complex problems are divided into smaller and simpler tasks**. These tasks are solved independently by functions.
- **A function is a block of organized, reusable code** that is used to perform a similar task of some kind.
- It **avoids the repetition of some codes over and over**. It means the programmer need not have to write same instructions for a number of times, to perform the same action.
- Functions help **easy debugging, testing and** understanding of the program.
- Functions **reduce program size and the program development time**.
- Functions provide better modularity and high degree of reusability for the problems.

NOTATIONS

- A notation is a system of characters, expressions, graphics or symbols used in problem solving process to represent technical facts to facilitate the best result for a problem.
- Example: Pseudocode, Flowchart.

Pseudocode

- ❑ Pseudocode is a **short, readable and formally styled** English language used for explaining an algorithm.
- ❑Pseudocode **does not include details like variable** declarations, subroutines etc. It is a short hand way of describing a computer program.
- ❑Using Pseudocode, it is **easier for a programmer or a non-programmer to understand** the general working of the program, because it is not based on any programming language.
- ❑It is used to **give a sketch of the structure of the program**, before the actual coding. It is **not machine readable.**
- ❑Pseudocode **cannot be complied or executed**. There is **no standard** for the syntax of Pseudocode.

Preparing a Pseudocode

- ❑Pseudocode is written using **structured English**.
- ❑The programmers may use their **own style for writing** the Pseudocode, which can be easily understood. **It never use the syntax** of any programming language.
- ❑Pseudocode uses some **keywords** to denote programming process. Some of them are as follows.

Input	**:** INPUT, GET, READ and PROMPT
Output	**:** OUTPUT, PRINT, DISPLAY and SHOW
Processing	**:** COMPUTE, CALCULATE, DETERMINE, ADD, SUBTRACT, MULTIPLY and DIVIDE
Initialize	**:** SET and INITIALISE
Incrementing	**:** INCREMENT

❑The Keywords should be capitalized.

❑There are three control structures used in Pseudocode.

❑They are,

a. Sequence control structure

b. Selection control structure and

c. Iteration control structure

Sequence Control Structures

- In sequence control structure, the steps are executed in a linear order one after the other. They are executed in the order in which they are written, from top to bottom.

EXAMPLE: Find product of any two numbers

```
READ values of A and B
COMPUTE C by multiplying A with B
PRINT the result C
STOP
```

Selection Control Structure

❑ In a selection control structure, the step to be executed next is based on a decision taken. If the condition is true, some path is followed. If the condition is false, different path is followed.

❑ There are two main selection structures: They are,

1. IF-THEN-ELSE statement
2. CASE statement

The IF-THEN-ELSE statement

- In IF-THEN-ELSE selection structure, if the condition is true, the THEN part is executed. Otherwise the ELSE part is executed.

```
IF condition THEN
            Process 1
ELSE
            Process 2
END IF
```

EXAMPLE: Find maximum of any three numbers

```
READ values of A, B, C
IF A is greater than B THEN
ASSIGN A to MAX
ELSE
ASSIGN B to MAX
IF MAX is greater than C THEN
PRINT MAX is greatest
ELSE
PRINT C is greatest
STOP
```

The case statement

- The case statement is used when many number of conditions to be checked. In a case statement, depending on the expression, one of the conditions is true. Based on the value, the corresponding statements are executed. If no match for the expression occurs, then the default option is executed.

```
CASE value1:
        Process1

CASE value2:
        Process2

CASE value n:
        Process n

END CASE
```

Iterative Control Structures

- In iterative control structure, a condition is checked. Based on the result of this conditional check (true or false) different paths are followed, where the control goes back to one of the already executed steps to make a loop.
- There are two iterative control structures. They are, WHILE and DO-WHILE.
- In WHILE loop, the condition is executed at the beginning of the loop. If the condition is false then the loop will not be executed.

```
WHILE condition
        Statements
END WHILE
```

```
DO
        Statements
WHILE condition
END DO
```

EXAMPLE: Find sum of first 100 integers

- INITIALIZE sum to zero
- INITIALIZE i to zero
- DO WHILE (i less than 100)
- INCREMENT i by 1
- ADD i to SUM and store in SUM
- END DO
- PRINT SUM
- STOP

Advantages of Pseudocode

- ❑ A Pseudocode is **closer to the programming code**. Thus it can be easily converted into the actual program.
- ❑Writing Pseudocode is much **easier** in comparison with drawing a flowchart.

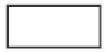
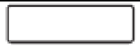
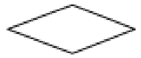
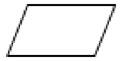
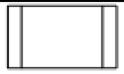
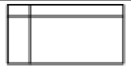
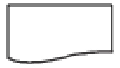
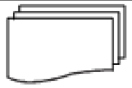
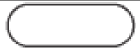
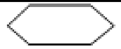
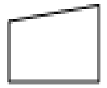
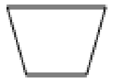
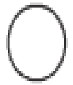
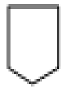
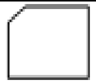
Limitations of Pseudocode

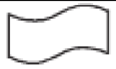
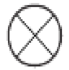
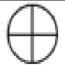
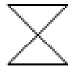
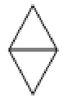
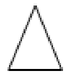
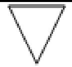
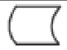
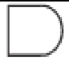
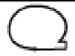
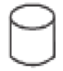
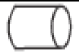
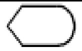
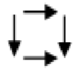
- ❑ It is **difficult to understand and manipulate** Pseudocode as compared to algorithm and flowchart.
- ❑As there are **no specific standards** for writing Pseudocode, it becomes hard to maintain consistency.

Flowchart

- ❑A flowchart is a diagrammatic representation of the logic for solving a task.
- ❑A flowchart is drawn using boxes of different shapes with lines connecting them to show the flow of control.
- ❑The purpose of drawing a flowchart is to make the logic of the program clearer in a visual form.
- ❑The logic of the program is communicated in a much better way using a flowchart. Since flowchart is a diagrammatic representation, it forms a common medium of communication.

Flowchart Symbols

Symbol	Symbol Name	Description
	Process	Operation or action step
	Alternate Process	Alternate to normal process
	Decision	Decision or a branch
	Data	Input/Output(I/O) to or form a process
	Predefined Process	Process Previously specified
	Internal storage	Stored in memory
	Document	A document
	Multi document	Multi document
	Terminator	Start or stop point
	Preparation	Set-up process
	Manual input	Data entry from a form
	Manual operation	Operation to be done manually
	Connector	Join flow lines
	Off-page connector	Continue on another page
	Card	I/O from a punched card

	Data	I/O to or from a process
	Summing junction	Logical AND
	OR	Logical OR
	Collate	Organize in a format
	Sort	Sort in some order
	Extract	Split process
	Merge	Merge in a predefined order
	Stored data	General data storage
	Delay	Wait
	Sequential access storage	Stored on magnetic taps
	Magnetic disk	I/O from magnetic disk
	Direct access storage	Storing on hard disk
	Display	Display output
	Flow lines	Indicates the direction of flow

Guidelines for preparing a flowchart

Certain rules that must be followed while drawing a flowchart are given below.

- ❑There can be only one start and one stop symbol in a flowchart.
- ❑Only one flow line can be used with the start and stop symbol.
- ❑Direction of flow of information in a flowchart must be from top to bottom or from left to right. Relevant symbols must be used while drawing a flowchart.
- ❑Only one flow line should come out from a process symbol.

- Only one flow line should come out from a process symbol.

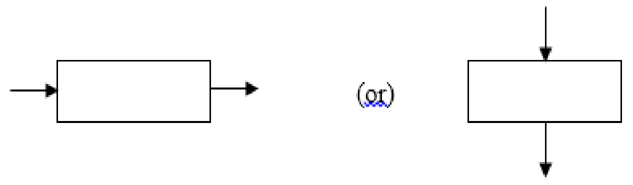


Fig. 1.2 *One Flow Line in Process*

- Only one flow line should enter a decision symbol, but two or three flow lines, one for each possible answer, should leave the decision symbol.

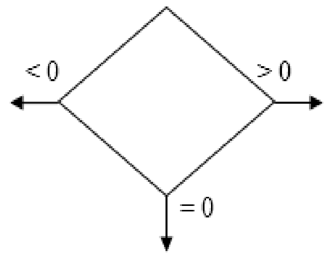
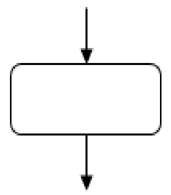


Fig. 1.3 *Multiple Flow Lines in a Decision*

- Only one flow line is used in conjunction with terminal symbol.

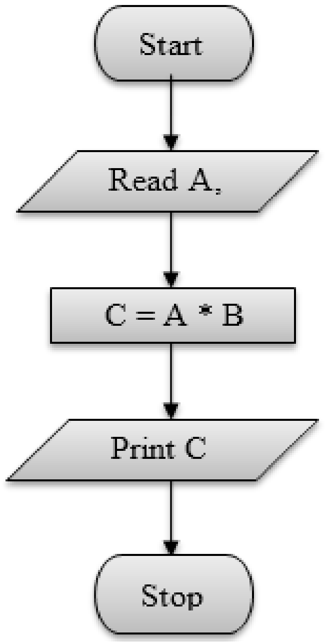


Flowchart

- ❑If the flowchart becomes complex, it is better to use connector symbols to reduce the number of flow lines.
- ❑The flow lines should not cross each other.
- ❑Ensure that the flowchart has a logical start and finish.
- ❑It is useful to test the validity of the flowchart by passing through it with a simple test data.

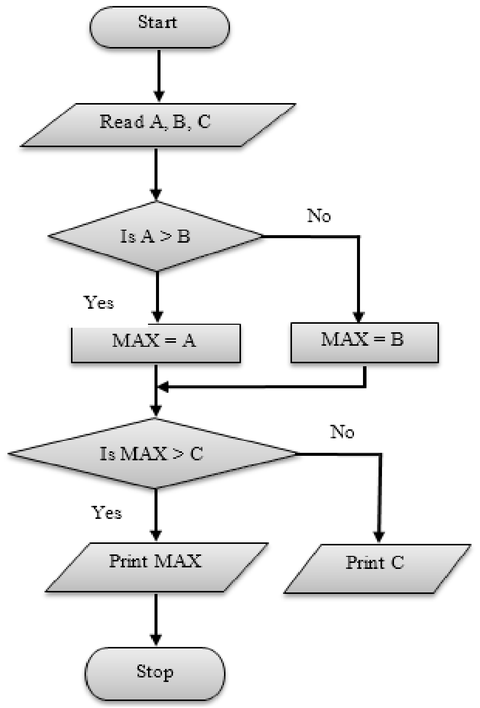
Sequence Control Structure

- In sequence control structure, the steps are executed in linear order one after the other.



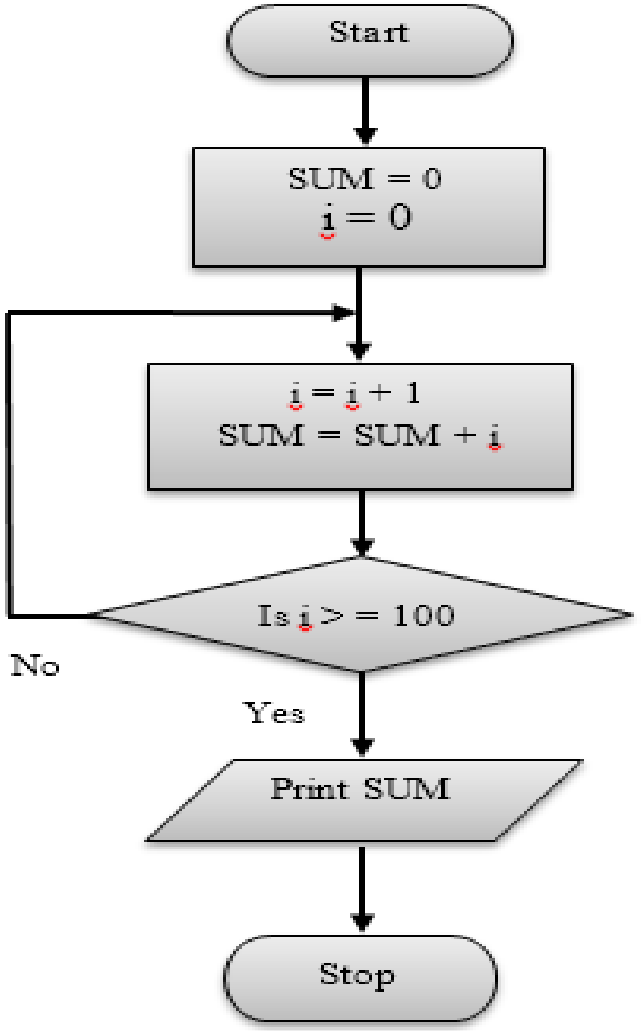
Selection Control Structure

- In a selection control structure, the step to be executed next is based on a decision taken. If the condition is true, one path is followed. If the condition is evaluated to false, another path is follo



Iterative Control Structure

- In iterative control structure, a condition is checked. Based upon the result of this conditional check (true or false) different paths are followed, where the control goes back to one of the already executed steps to make a loop.



Advantages of flowcharts

- ❑The symbols used in a flowchart are self-explanatory. This makes a flowchart easier to understand.
- ❑Being a graphical representation, flowcharts better communicate the problem solving logic to the users.
- ❑With the help of a flowchart, a problem can be analysed in an effective way.
- ❑It also helps in monitoring, data collection and identifies areas for improvement or increase in efficiency.

Limitations of flowchart

- ❑In case of large programs, the flowcharts may continue too many pages. This makes them difficult to understand.
- ❑It takes a lot of time to represent a program diagrammatically.
- ❑Any changes or modification in a flowchart usually requires redrawing the entire flowchart. It is a challenging job.

Programming Language

- ❑ A programming language is a simplified form of English (with math symbols) that adheres to a strict set of grammatical rules.
- ❑Programming languages, limit the vocabulary and grammar to become much simpler. Although a programming language is simple in form, it is not always easy to use.
- ❑Programming languages forces the user to write very simple, exact instructions.

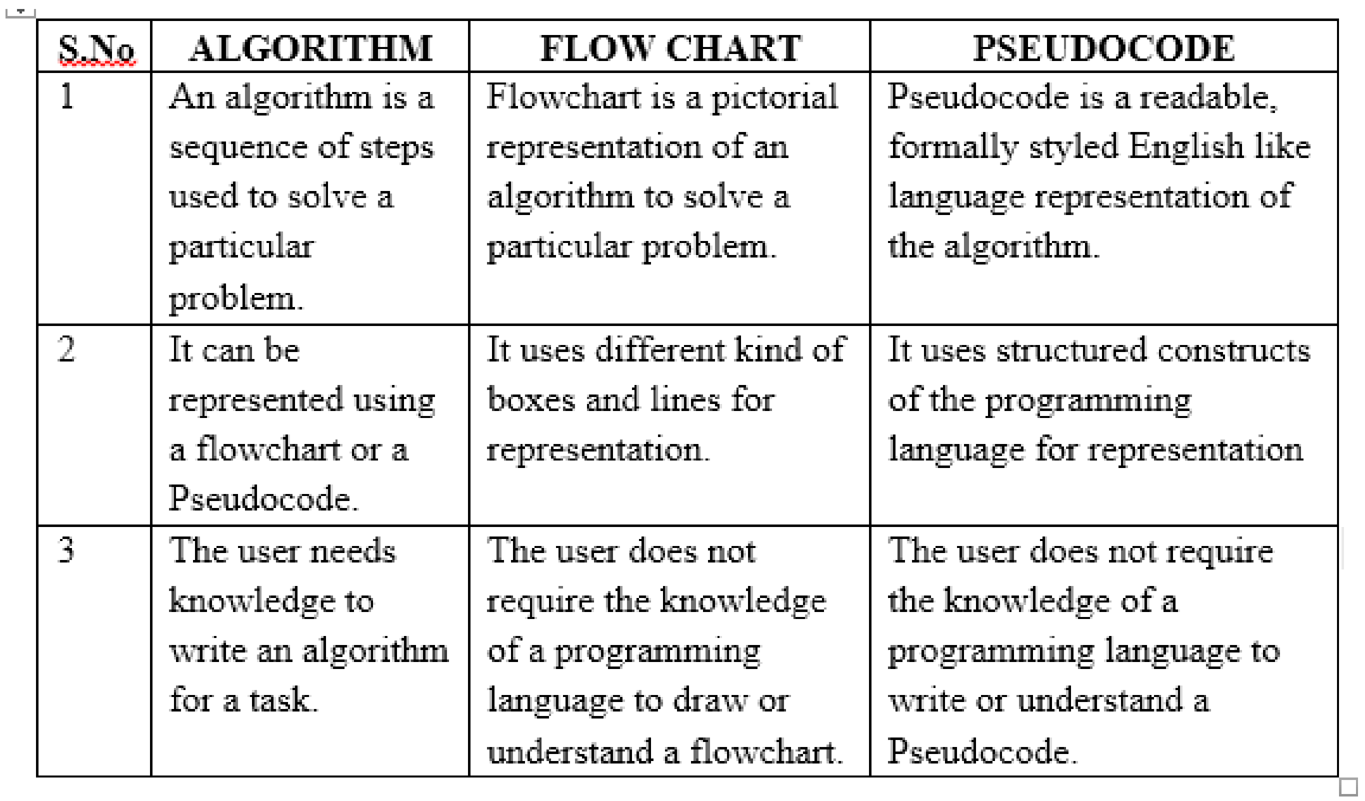
Coding

- ❑Translating an algorithm into a programming language is called coding the algorithm.
- ❑The product of the translation or the coding for all the algorithms in the problem are tested by collecting them into programs and executing them on computers.
- ❑If the program fails to produce the desired results, the programmer must debug it and determine what is wrong and then modify the program.

Implementation

- ❑The combination of coding and testing algorithms is called implementation.
- ❑Code is the product of translating an algorithm into a programming language. The term code may refer to a complete program or to any portion of a program.

Difference between Algorithm, Flowchart, and Pseudocode



S.No	ALGORITHM	FLOW CHART	PSEUDOCODE
1	An algorithm is a sequence of steps used to solve a particular problem.	Flowchart is a pictorial representation of an algorithm to solve a particular problem.	Pseudocode is a readable, formally styled English like language representation of the algorithm.
2	It can be represented using a flowchart or a Pseudocode.	It uses different kind of boxes and lines for representation.	It uses structured constructs of the programming language for representation
3	The user needs knowledge to write an algorithm for a task.	The user does not require the knowledge of a programming language to draw or understand a flowchart.	The user does not require the knowledge of a programming language to write or understand a Pseudocode.

ALGORITHMIC PROBLEM SOLVING

- ❑Every problem's solution starts with a plan and that plan is called an algorithm. So, an algorithm is a plan for solving a problem.
- ❑There are many ways to write an algorithm. Some are very informal, some are quite formal and mathematical in nature, and some are quite graphical.
- ❑The development of an algorithm is the key step in algorithmic problem solving. Once an algorithm is developed, it can be translated into a computer program in some programming language.

Algorithmic Problem Solving

The algorithmic problem solving process consists of five major steps.

1. Obtain a description of the problem.
2. Analyze the problem.
3. Develop a high-level algorithm.
4. Refine the algorithm by adding more details.
5. Review the algorithm.

Step 1: Obtain a description of the problem

- The algorithmic problem solving process starts by obtaining a description of the problem. This step is much more difficult than it appears. In the software development process, the word ‘client’ refers to a person who wants to find a solution to a problem, and the word ‘developer’ refers to a person who finds a way to solve the problem. The developer must create an algorithm that will solve the client's problem.
- The client is responsible for creating a description of the problem, but this is often the weakest part of the process. A problem description suffers from one or more of the following types of defects:
  - the description may rely on unstated assumptions
  - the description may be ambiguous
  - the description maybe incomplete
  - the description may have internal contradictions
- These defects are due to carelessness by the client. Part of the developer’s responsibility is to identify, the defects in the description of a problem, and to work with the client to rectify those defects.

Step 2: Analyze the problem

- ❑The purpose of this step is to determine both the starting and the ending points for solving the problem.
- ❑This process is analogous to a mathematician determining what is given, and what must be proven. A good problem description makes it easier to perform this step.
- ❑ Asking the following questions often helps to determine the starting point.
  - ✓ What data are available?
  - ✓Where is that data from?
  - ✓What are the rules exist for working with the data?
  - ✓What are the relationships exist among the data values?

Step 2: Analyze the problem

- ❑ When determining the ending point, algorithm need to describe the characteristics of a solution. Asking the following questions often helps to determine the ending point.
  - What new facts will we have?
  - What items would have changed?
  - What changes would have been made to those items?
  - What things will no longer exist?

Step 3: Develop a high-level algorithm

- ❑ An algorithm is a plan for solving a problem, but plans come in several levels of detail.
- ❑It is better to start with a high-level algorithm that includes the major part of a solution, but leaves the details until later.

Step 4: Refine the algorithm by adding more detail

- ❑ A high-level algorithm shows the major steps that need to be followed to solve a problem. Our goal is to develop algorithms that will lead to computer programs.
- ❑Consider the capabilities of the computer and provide enough details so that someone else could use the algorithm to write the computer program.
- ❑For larger, more complex problems, it is common to go through this process several times. Each time, more details are added to the previous algorithm, for further refinement.
- ❑This technique of gradually working from a high-level to a detailed algorithm is often called stepwise refinement.
- ❑Stepwise refinement is a process of developing a detailed algorithm by gradually adding details to a high-level algorithm.

Step 5: Review the algorithm

- ❑ The final step is to review the algorithm. First, work through the algorithm step by step to determine whether or not, it will solve the original problem. Asking these questions and seeking their answers is a good way to develop skills that can be applied to the next problem.
- ❑ Does this algorithm solve a very specific problem or does it solve a more general problem?
- ❑ If it solves a very specific problem, should it be generalized?
- ❑ For example, an algorithm that computes the area of a circle having radius 5.2 meters (formula $\pi*5.22$) solves a very specific problem, but an algorithm that computes the area of any circle (formula $\pi \times r^2$) solves a more general problem.

Examples

- Can this algorithm be simplified?
  - For example, one formula for computing the perimeter of a rectangle is:
  - length + width + length + width
  - A simpler formula would be:
  - 2.0* (length + width)
- Is this solution similar to the solution to another problem? How are they alike? How are they different?
  - For example, consider the following two formulae:
  - Area of_a Rectangle = length * width
  - Area_of_a Triangle = 0.5 * base * height
-

Additional points

- **Similarities:** Each computes an area. Each multiplies two measurements.
- **Differences:** Different measurements are used. The triangle formula contains 0.5.
- **Hypothesis:** Perhaps, every area formula involves multiplying two measurements.
- Once an algorithm is developed, translate it into a computer program in some programming language.

SIMPLE STRATEGIES FOR DEVELOPING ALGORITHMS

Iteration

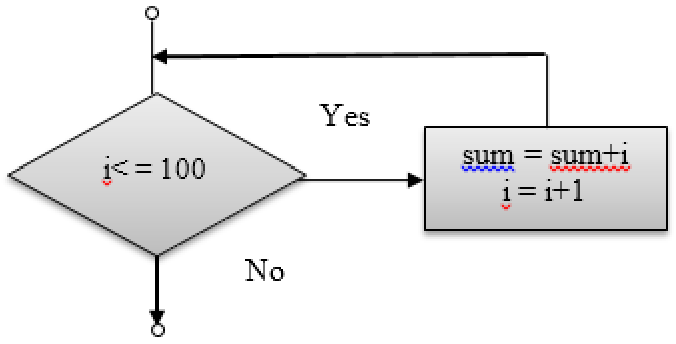
- Iteration is a process of repeating the same set of statements again and again until the specified condition holds true. Humans find iterative tasks boring but computers are very good at performing iterative tasks. Computers execute the same set of statements again and again by putting them in a loop.
- In general, loops are classified as:
  1. Counter-controlled loops
  2. Sentinel-controlled loops

Counter-Controlled Loops

- ❑Counter-controlled looping is a form of looping in which the number of iterations to be performed is known in advance.
- ❑Counter-controlled loops are so named because they use a control variable, known as the loop counter, to keep a track of loop iterations.
- ❑The counter-controlled loop starts with the initial value of the loop counter and terminates when the final value of the loop counter is reached.
- ❑Since the counter-controlled loops iterate a fixed number of times, which is known in advance, they are also known as definite repetition loops.

Example: Consider a program segment to find the sum of first 100 integers.

```
i = 1
sum = 0
while (i<= 100):
        sum = sum + i
         i = i + 1
```



Sentinel-Controlled Loops

- ❑In sentinel-controlled looping, the number of times the iteration is to be performed is not known beforehand.
- ❑The execution or termination of the loop depends upon a special value called the sentinel value.
- ❑If the sentinel value is true, the loop body will be executed, otherwise it will not.
- ❑Since the number of times a loop will iterate is not known in advance, this type of loop is also known as indefinite repetition loop.

Algorithm: To find the sum of first N integers

1. Start
2. Read N
3. Assign sum = 0, i = 0
4. Calculate i = i + 1 and sum = sum + i
5. Check whether i>= N, if no repeat step 4. Otherwise go to next step.
6. Print the value of sum
7. Stop

Recursion

- ❑Recursion is a powerful programming technique that can be used to solve the problems that can be expressed in terms of similar problems of smaller size.
- ❑For example, consider a problems to find the factorial of a number n. The problem of finding the factorial of n can be expressed in terms of a similar problem of smaller size as $n! = n \times (n-1)!$. Recursion provides an elegant way of solving such problems.
- ❑ In recursive programming, a function calls itself. A function that calls itself is known as a recursive function, and the phenomenon is known as recursion.

Recursion

Recursion is classified according to the following criteria:

1. Whether the function calls itself directly (i.e. direct recursion) or indirectly (i.e. indirect recursion).
2. Whether there is any pending operation on return from a recursive call. If the recursive call is the last operation of a function, the recursion is known as tail recursion.

A Function is directly recursive if it calls itself, i.e. the function body contains an explicit call to itself.

Indirect recursion occurs when a function calls another function, which in turn calls another function, eventually resulting in the original function being called again.

Example: Recursive algorithm for finding the factorial of a number

```
Step 1: Start
Step 2: Read number n
Step 3: Call factorial (n)
Step 4: Print factorial f
Step 5: Stop
```

```
Factorial (n)
 Step 1: if n = = 1 then return 1
Step 2: Else
            f = n*factorial (n-1)
Step 3: Return f
```